

Projekt końcowy

Adversarial Attacks

**Magdalena Borkowska
Marcin Świerczyński
Sławomir Ossowski
Kacper Jakubaszek
Michael Bartyzel**

Roadmap

1. Sieci neuronowe i konwolucyjne

3. Fast Gradient Sign Method

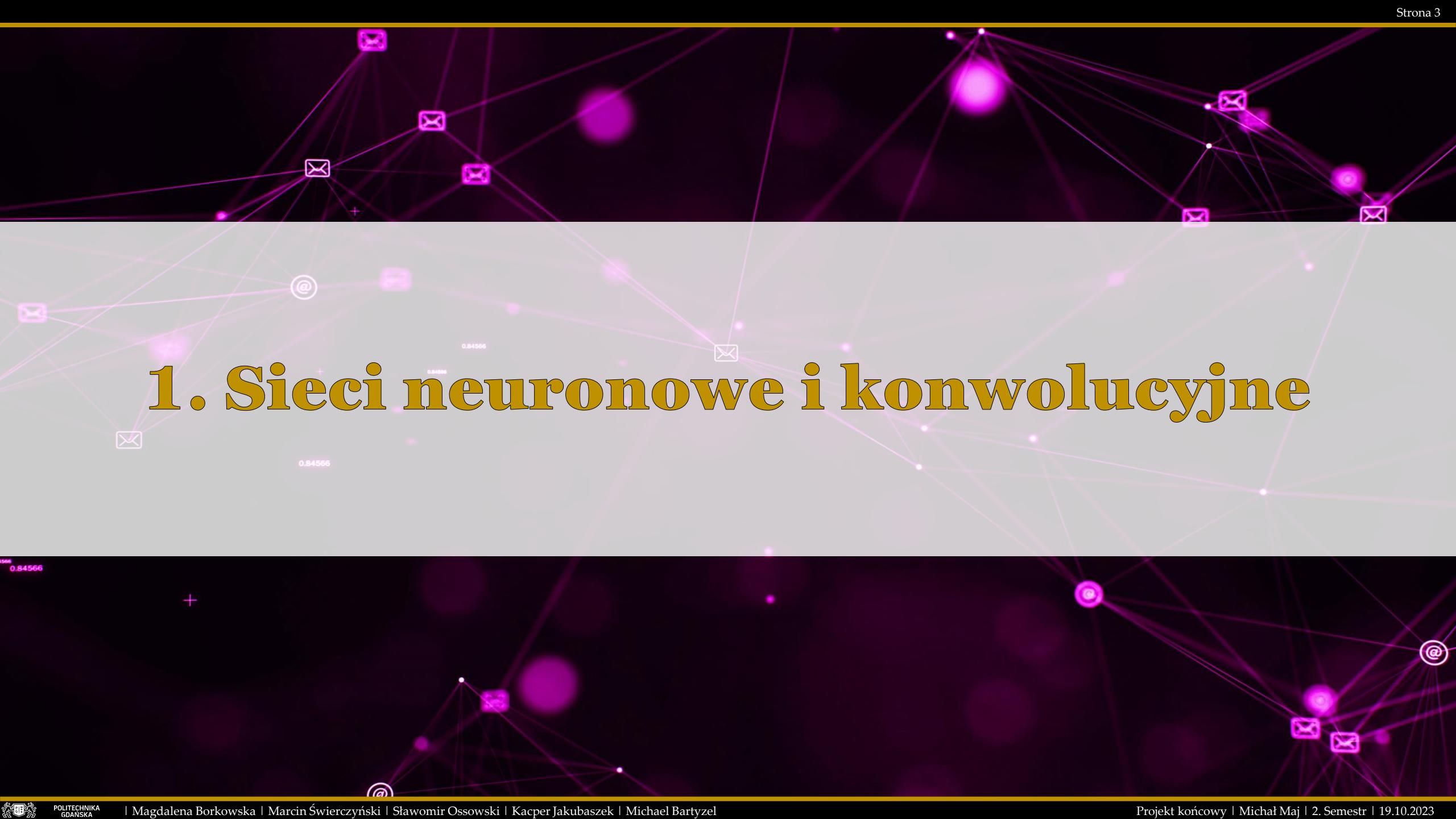
2. Ataki adwersarialne

4. Zbiór danych

5. Model

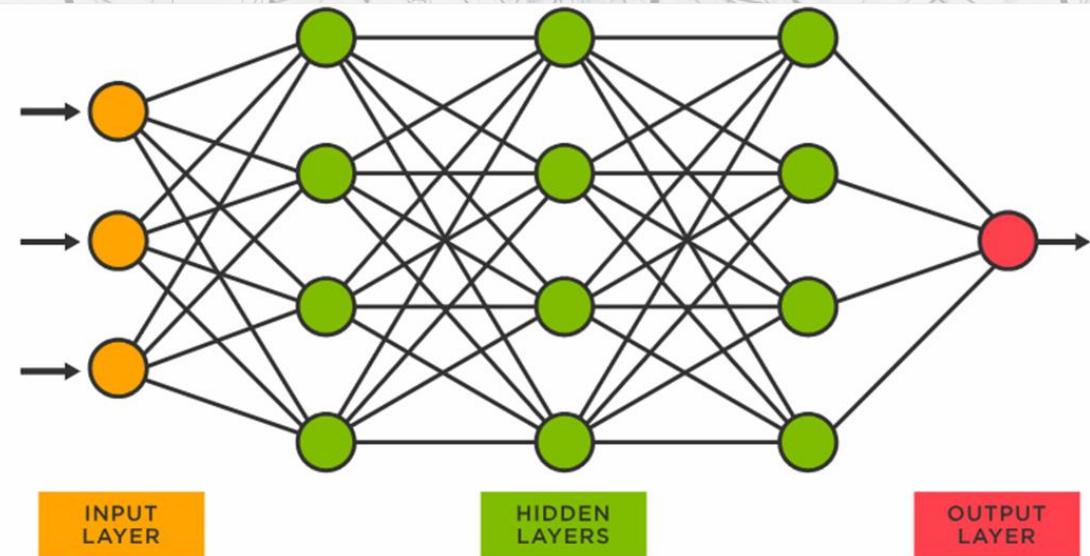
6. Wnioski

1. Sieci neuronowe i konwolucyjne



- Sieci neuronowe są rodzajem algorytmów uczenia maszynowego, które naśladują działanie ludzkiego mózgu.
- Są one zbudowane z połączonych ze sobą sztucznych neuronów, które przetwarzają i analizują dane, aby wykonywać określone zadania.
- Podstawowym elementem sieci neuronowych jest neuron, który otrzymuje pewne dane wejściowe, przetwarza je i generuje odpowiedź.
- Neurony są ze sobą połączone za pomocą wag, które określają siłę połączenia między neuronami.
- Sieć neuronowa składa się z warstw neuronów, które są ułożone w określonych strukturach.
- Sieci neuronowe mogą być wykorzystywane do różnych zadań, takich jak rozpoznawanie obrazów, klasyfikacja danych, rozpoznawanie mowy, predykcja trendów rynkowych, tłumaczenie języka, automatyczne sterowanie pojazdami, generowanie muzyki i wiele innych.
- Mogą być również wykorzystywane do rozwiązywania problemów, które są trudne do uchwycenia za pomocą tradycyjnych algorytmów programowania.
- Dzięki zdolności sieci neuronowych do uczenia się na podstawie danych, są one szczególnie przydatne w sytuacjach, w których istnieje duża ilość danych treningowych, a zadanie polega na wykryciu wzorców lub zależności w tych danych.
- Sieci neuronowe potrafią adaptować się do nowych danych i poprawiać swoje wyniki w miarę zdobywania większego doświadczenia.

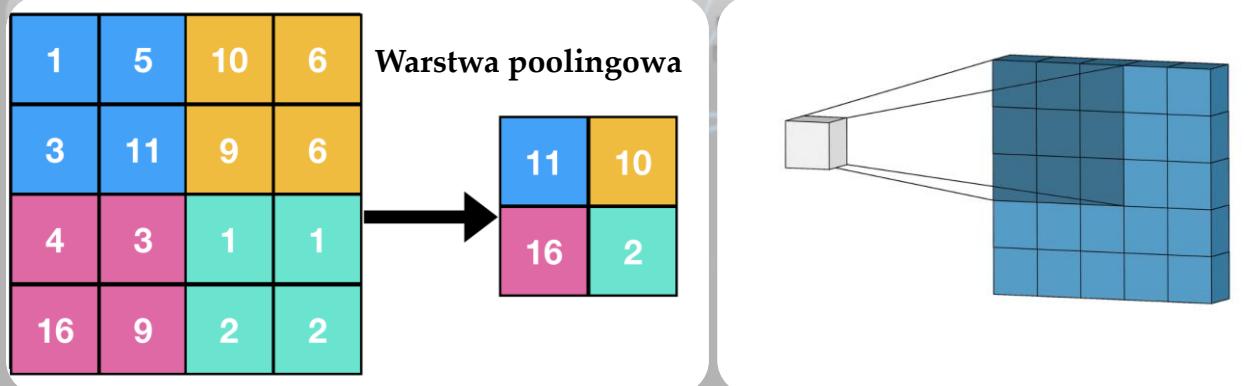
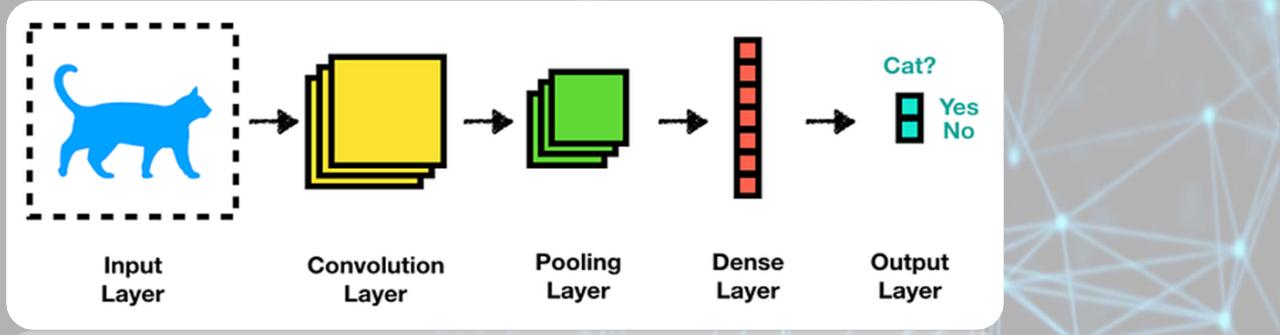
Sieci neuronowe



Źródła:

- www.projectpro.io/article/neural-network-projects/440
- www.tibco.com/reference-center/what-is-a-neural-network
- www.unite.ai/liquid-neural-networks-definition-applications-challenges/

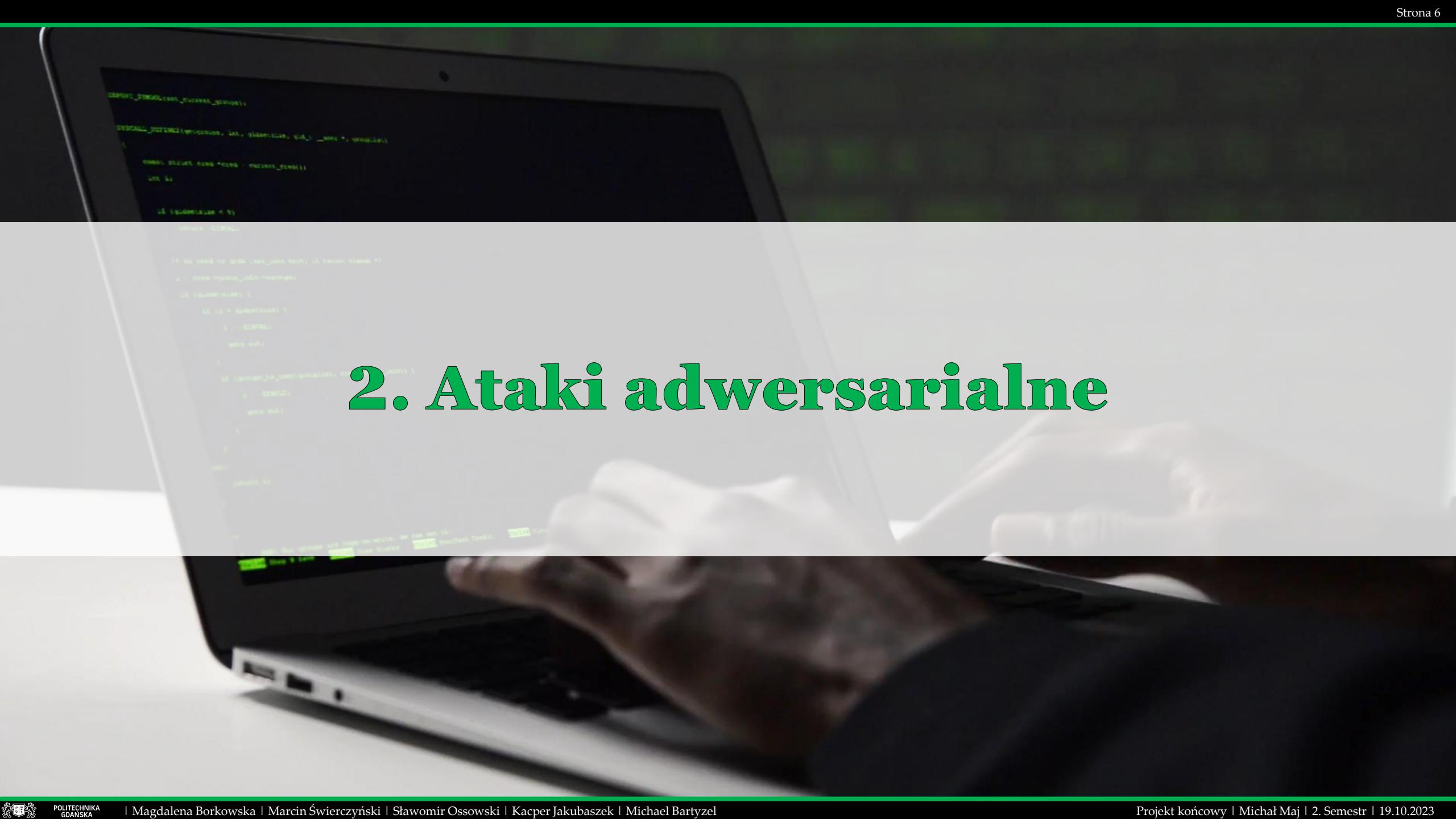
Sieci konwolucyjne

**Źródła:**

- www.towardsdatascience.com/convolutional-neural-network-a-step-by-step-guide-a8b4c88d6943

- Sieci konwolucyjne (Convolutional Neural Networks - CNN), to rodzaj algorytmu głębokiego uczenia, który jest najczęściej stosowany do analizy i poznawania cech wizualnych w oparciu o duże ilości danych
- Chociaż CNN są szeroko stosowane w dziedzinach związanych z przetwarzaniem obrazów i wizją komputerową, to mogą być używane do innych zadań zakresu AI, takich jak m.in. przetwarzanie języka naturalnego
- Konwolucyjne sieci neuronowe działają poprzez pozyskiwanie i przetwarzanie dużych ilości danych w formacie siatki, a następnie wyodrębnianie istotnych cech szczegółowych w celu klasyfikacji i wykrywania.
- CNN składają się zwykle z trzech rodzajów warstw: warstwy konwolucyjnej, warstwy buforowej oraz warstwy w pełni połączonej.
- Każda warstwa pełni inną rolę, wykonuje zadanie na pozyskanych danych i uczy się coraz bardziej skomplikowanych rzeczy.
- Podstawowym elementem sieci konwolucyjnych jest warstwa konwolucyjna, która wykonuje operację konwolucji na danych wejściowych.
- Operacja konwolucji polega na przesuwaniu filtru (takiego jak macierz wag) po wejściowej macierzy danych, obliczając ważoną sumę wartości dla każdej lokalnej części danych.
- Ta operacja pozwala sieci na wykrywanie lokalnych wzorców i cech w danych, takich jak krawędzie, tekstury i inne istotne detale.
- Warstwa poolingowa (najczęściej warstwa MaxPooling) zmniejsza rozmiar danych poprzez zastępowanie lokalnych grup wartości największą wartością w grupie.
- Pozwala to na zmniejszenie liczby parametrów i obliczeń w sieci, jednocześnie zachowując ważne informacje o lokalnych cechach.
- Sieci konwolucyjne są zazwyczaj zbudowane z wielu warstw konwolucyjnych i poolingowych, które są odpowiedzialne za wykrywanie coraz bardziej skomplikowanych cech w danych.
- Następnie wykorzystuje się warstwy w pełni połączone, które agregują wykryte cechy i wykonują końcową klasyfikację lub prognozę.

2. Ataki adwersarialne



- Cyberataki przeprowadzane na systemach wykorzystujących uczenie maszynowe.
- Polegają na celowym oszukaniu modelu przy wykorzystaniu jego słabych stron.
- Zakłócenia wprowadzane są do danych wejściowych aby zdezorientować model, doprowadzić do obniżenia jego dokładności i wydajności.
- Celem ataków adwersarialnych może być obejście zabezpieczeń systemu, oszukanie modelu, wprowadzenie dezinformacji lub wykorzystanie systemu w sposób niezamierzony przez twórców.
- Badanie i rozwój metod obrony przed atakami adwersarialnymi stanowi ważne zagadnienie w dziedzinie uczenia maszynowego, aby zapewnić większą odporność i niezawodność systemów uczących na tego rodzaju manipulacje.
- Najczęściej stosowanym rodzajem ataku adwersarialnego jest atak zakłóceń, w którym dokonuje się minimalnych zmian w danych wejściowych, które są trudne do wykrycia przez człowieka, ale wystarczające, aby wprowadzić znaczące zmiany w rezultatach modelu.
- Te zmiany mogą prowadzić do błędnych klasyfikacji obiektów, przekłamanych wyników lub zmiany działania systemu w nieprzewidywalny sposób.
- Techniki:** Poisoning, Evasion, Transferability, Surrogacy
- Ochrona:** Adversarial Training, Regular Auditing, Data Sanitization, Security Updates

Ataki adwersarialne

Formy:

1) Dostęp atakującego do początkowych parametrów modelu:

- White-box** - Atakujący jest całkowicie świadomym parametrów docelowego modelu (jego architektury, funkcji straty, danych treningowych, itp.)
- Black-box** – Atakujący nie posiada żadnych wiadomości o atakowanym modelu

2) Metoda tworzenia obrazu adwersarialnego:

- Non-targeted** - Atakujący przypisuje obraz podlegający atakowi do dowolnej klasy, niezależnie od klasy prawdziwego obrazu,
- Targeted** - Atakujący przypisuje obraz podlegający atakowi do określonej klasy.

Źródła:

- <https://www.makeuseof.com/what-are-adversarial-attacks-ai-models-and-how-to-stop-them/>

3. Fast Gradient Sign Method

FGSM

Fast Gradient Sign Method

Przykład:

$$x = x_0 - \eta \cdot \text{sign}(w)$$

nie specyfikowany

Źródła:

- <https://engineering.purdue.edu/ChanGroup/ECE595/files/chapter3.pdf>
- https://www.tensorflow.org/tutorials/generative/adversarial_fgsm

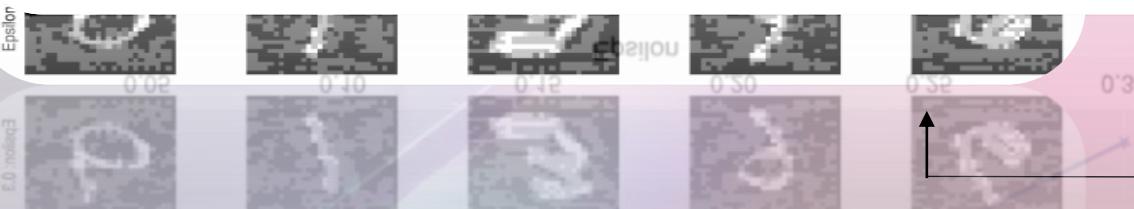
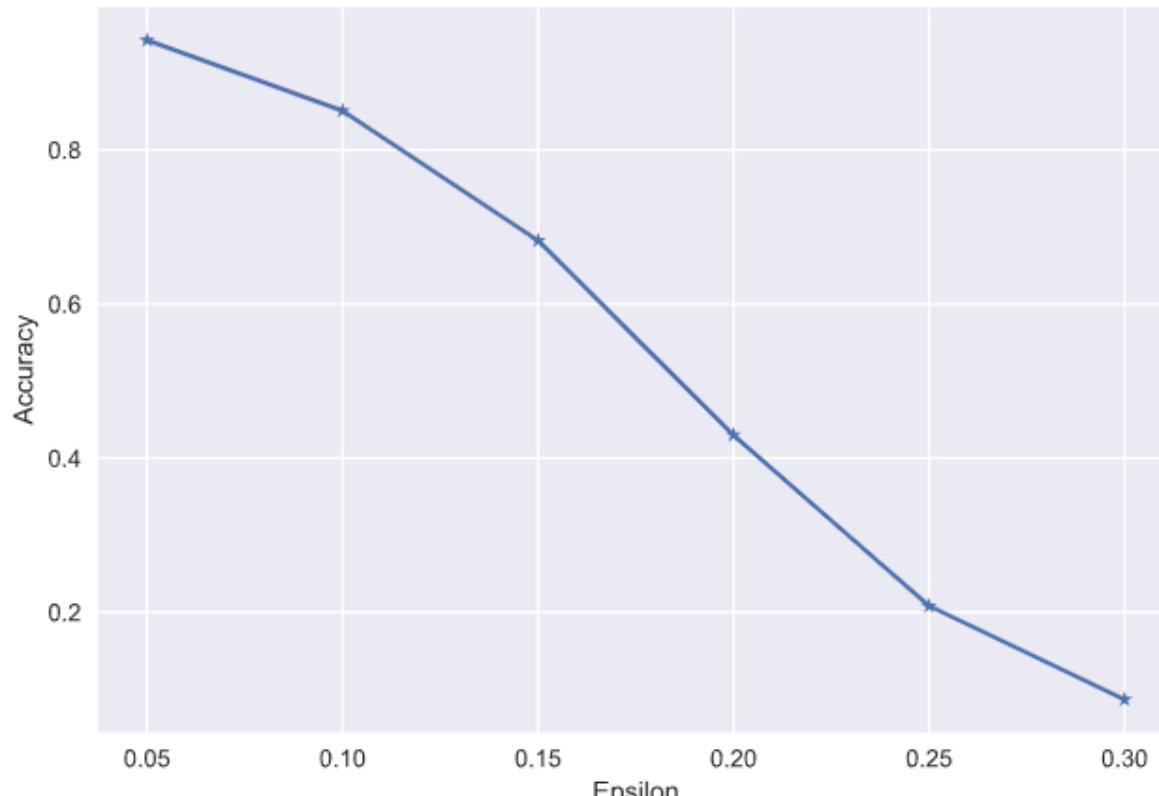
Opis:

- Pochodna funkcji straty
- Metoda jest szybka ponieważ pochodna jest łatwa do zrealizowania, nawet w głębokich sieciach neuronowych przez tzw. back propagation, które jest dostępne na większości platformach programowania
- Kierunek wymiaru zaburzenia ℓ_∞ zależy od znaku pozytywnego lub negatywnego pochodnej
- Przekształcona forma ataków na głębokie sieci neuronowe
- Maksymalizacja pewnej funkcji straty $J(x; w)$
- Funkcja straty $J(x; w)$ posiada klasyfikator (parametr) w
- Przez maksymalizowanie straty pogorszamy parametr w
- Przy funkcji straty $J(x; w)$ FGSM tworzy atak x :
 - $x = x_0 + \eta \cdot \text{sign}(\nabla_x J(x_0; w))$
- Funkcja jest zależna od x a nie od w , dlatego $\nabla_x J(x_0; w)$ jest pochodną od J w zależności od x
- Zazwyczaj: Funkcja straty = Strata treningu
- FGSM oblicza pochodną funkcji straty (np. mean-squared error, categorical cross-entropy) na podstawie zdjęcia aby kreować nowe zdjęcie (np. adversarial image) które maksymalizuje stratę
- FGSM używa znaku aby stworzyć modyfikacje do inputu które się zbiera do pewnej misklasyfikacji, ale nadal zostaje "dość małe"
- Używanie pełnej informacji pochodnej kreuje większą zmianę dla inputu które nie usatysfakcjonuje ograniczenie $\|\eta\|_\infty$

Metoda:

- Wczytać input zdjęcia
- Zrobić predykcje na podstawie zdjęcia używając ztrenowanego CNN
- Wyliczyć stratę predykcji na podstawie prawdziwej etykiety klasy
- Wyliczyć gradient (nachylenie) straty na podstawie inputu zdjęcia
- Wyliczyć znak (sign) gradiента (nachylenia)
- Używać gradiента ze znakiem aby stworzyć output adwersarialnego zdjęcia

FGSM



Przykłady – Zmienianie liczb [1/2]

```
def fgsm_attack(image, epsilon):
    perturbed_image = image + epsilon * image.grad.data.sign()
    perturbed_image = torch.clamp(perturbed_image, 0, 1)
    return perturbed_image

def test(model, test_loader, epsilon, device, num_display=5):
    num_correct = 0
    adv_examples = []

    for image, label in test_loader:
        image = image.to(device)
        label = label.to(device)
        image.requires_grad = True

        output = model(image)
        _, init_pred = output.max(dim=1)

        if init_pred.item() != label.item():
            continue

        loss = F.nll_loss(output, label)
        model.zero_grad()
        loss.backward()

        perturbed_image = fgsm_attack(image, epsilon)
        perturbed_output = model(perturbed_image)
        _, perturbed_pred = perturbed_output.max(dim=1)

        if perturbed_pred.item() == label.item():
            num_correct += 1
        elif len(adv_examples) < num_display:
            adv_examples.append(
                (
                    label.item(),
                    perturbed_pred.item(),
                    perturbed_image.squeeze(0).detach().cpu().numpy()
                )
            )

    accuracy = num_correct / len(test_loader)
    print(f'Epsilon: {epsilon}, Accuracy: {accuracy:.3f}')

    return accuracy, adv_examples
num_display = 5
accuracies = []
all_adv_examples = []
epsilons = [.05, .1, .15, .2, .25, .3]

for epsilon in epsilons:
    accuracy, adv_examples = test(model, test_loader, epsilon, device, num_display)
    accuracies.append(accuracy)
    all_adv_examples.append(adv_examples)
```

Epsilon: 0.05, Accuracy: 0.943
Epsilon: 0.1, Accuracy: 0.851
Epsilon: 0.15, Accuracy: 0.683
Epsilon: 0.2, Accuracy: 0.430
Epsilon: 0.25, Accuracy: 0.208
Epsilon: 0.3, Accuracy: 0.087

```
plt.style.use("seaborn")
plt.plot(epsilons, accuracies, "-.")
plt.xlabel("Epsilon")
plt.ylabel("Accuracy")
plt.show()

def clean_axis(axis):
    axis.set_xticks([])
    axis.set_yticks([])
    axis.set_xticklabels([])
    axis.set_yticklabels([])

nrows = len(epsilons)
ncols = num_display

fig, ax = plt.subplots(6, 5, figsize=(10, 10))

for row in range(nrows):
    for col in range(ncols):
        label, perturbed_pred, perturbed_image = all_adv_examples[row][col]
        axis = ax[row, col]
        axis.imshow(perturbed_image, cmap="gray")
        axis.set_title(f'{label} -> {perturbed_pred}')
        clean_axis(axis)
        if col == 0:
            axis.set_ylabel(f'Epsilon: {epsilons[row]}')

plt.tight_layout()
plt.show()
```

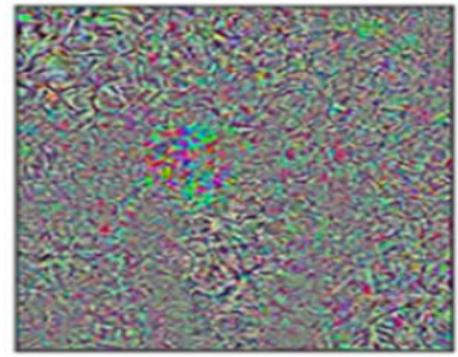
źródła:
<https://jaketae.github.io/study/fgsm/>

FGSM

Przykłady – Generowanie adwersarialnych zdjęć [2/2]

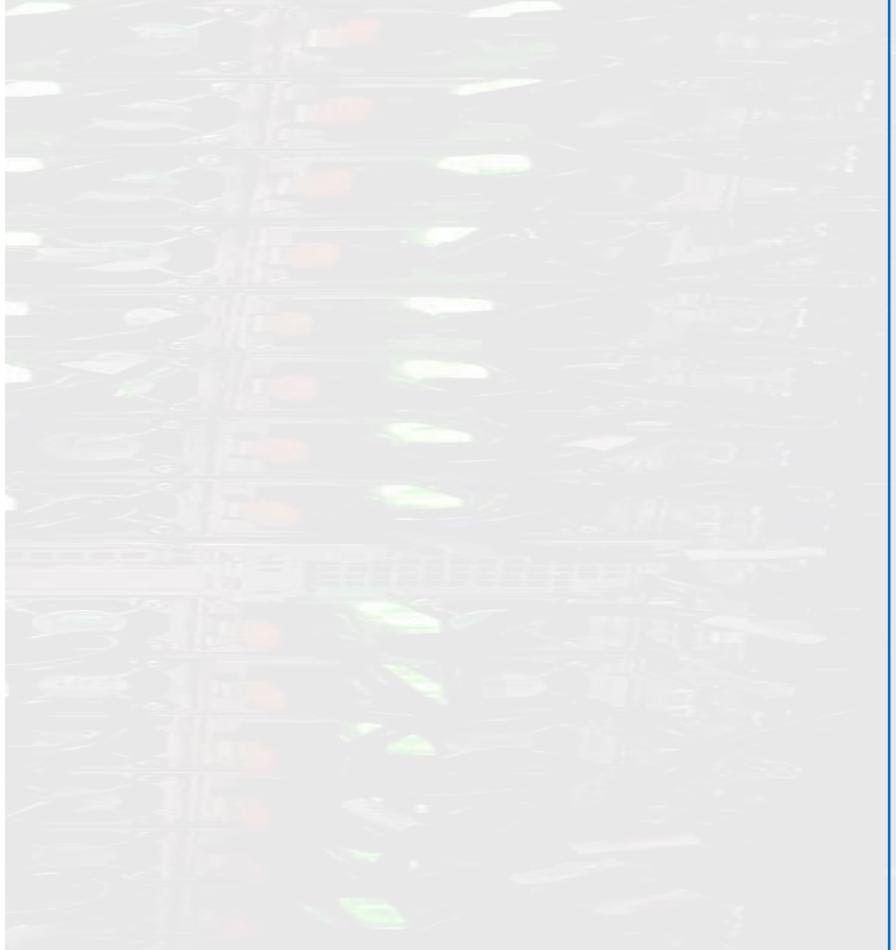
 X

97.3% macaw

 $\text{sign}(\nabla_x J(\theta, X, Y))$ $=$  $X + \epsilon \cdot \text{sign}(\nabla_x J(\theta, X, Y))$
88.9% bookcase x

“panda”

57.7% confidence

 $\text{sign}(\nabla_x J(\theta, x, y))$
“nematode”
8.2% confidence $=$  $x + \epsilon \text{sign}(\nabla_x J(\theta, x, y))$
“gibbon”
99.3 % confidence

Źródła:

❑ https://pytorch.org/tutorials/beginner/fgsm_tutorial.html

4. Zbiór danych

Dane Alien vs Predator

Rozpoznawanie i odróżnianie zdjęć:

- Computer Vision
- Klasyfikacja zdjęć – 2 klasy
- Kategoryzacja zdjęć – 2 kategorie
- Odróżnianie zdjęć za pomocą pikseli
- Głębokie sieci neuronowe w Pythonie używają bibliotek Keras oraz TensorFlow

- Format: obrazy JPG , różne rozmiary miniatur (około 250 x 250 px)
- Zbiór treningowy - 347 aliens i 347 predators
- Zbiór testowy - 9 aliens i 9 predators
- Zbiór walidacyjny - 91 aliens i 91 predators

Podejście:

- 1) Wczytać dane
- 2) Definiować prostą sieć konwolucyjną
- 3) Trenować model
- 4) Oceniać dokładność tego modelu na podstawie danych testowych
- 5) Pokazać model do przewidywania rodzaju zwierzęcia na podstawie obrazu

**Parametry:**

- System operacyjny Windows 10
- Procesor Intel Celeron 2GHz
- Pamięć DDR4, 16 GB
- Płyta główna ASRock H110 Pro

Opis:

- 13 x GPU Nvidia GTX 1660 Super (8 GB RAM)
- Używana do kopania kryptowaluty
- Platforma Multi-GPU (równoległość, efektywność i przetwarzanie bardziej zaawansowanych modeli)

Zastosowanie:

- Wielokartowe platformy do trenowania AI przez GPU / TPU (karta przetwarzania tensorowego)
- Przewaga wykorzystania wielu GPU (Big Data i skomplikowane obliczenia, równolegle szybsze przetwarzanie)
- Wielokartowe platformy w głębokim uczeniu (w CNN i rozpoznawaniu obrazów bardzo efektywne)
- Frameworki do trenowania z wieloma GPU (TensorFlow, PyTorch, MXNet do skalowania modelu)

Przetwarzanie danych

Parallelism:

- Równoczesne obliczenia dla lepszej wydajności i przyspieszenia trenowania modelu przez dane/ operacje szczególnie w CNN, Machine Learning i AI

Formy paralelizmu:

- Data Parallelism
- Model Parallelism
- Task Parallelism
- Algorithm-level Parallelism

Optymalizacja (ulepszać wyniki i proces trenowania):

- Wybór architektury modelu która odciąży zasoby
- Techniki regularyzacji (dropout, L2 regularization) aby nie przeuczać modelu i odciążić RAM
- Hiperparametry modelu przy mniejszych zasobach
- Kompresja modeli (zmniejszanie rozmiaru przy tej samej wydajności)
- Zaawansowane techniki propagacji wstecznej (Back Propagation) przez optymalizatory Adam/ RMSprop

html

```
    $('.count').each(function () {  
        $(this).prop('Counter', 0).animate({  
            Counter: $(this).text()  
        }, {  
            duration: 4000,  
            easing: 'swing',  
            step: function (now) {  
                $(this).text(Math.ceil(now));  
            }  
        });  
    });
```

5. Model

```
function() {  
    var objeto = document.getElementById('objeto');  
    objeto.onclick = function() {  
        //...  
    };  
};
```



Logout

jupyter FGSM-alien-trained Last Checkpoint: W zeszły czwartek o 19:04 (autosaved)

File Edit View Insert Cell Kernel Help

Not Trusted

Python 3 (ipykernel) 

```
In [1]: # Na podstawie https://www.tensorflow.org/tutorials/generative/adversarial_fgsm
```

```
In [1]: train_dir = "data/alien-vs-predator/train/"
validation_dir = "data/alien-vs-predator/validation/"
test_dir = "data/alien-vs-predator/test/"
```

```
In [2]: import pandas as pd
import tensorflow as tf
import numpy as np
import random
import os
import matplotlib as mpl
import matplotlib.pyplot as plt

mpl.rcParams['figure.figsize'] = (8, 8)
mpl.rcParams['axes.grid'] = False
plt.rcParams['figure.figsize'] = [20, 10]

tf.__version__
```

```
Out[2]: '2.13.0'
```

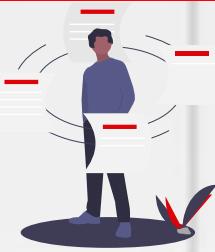
```
In [3]: model = tf.keras.models.load_model("models/alien_vs_predator_FGSM.h5")
```

WARNING:absl:At this time, the v2.11+ optimizer `tf.keras.optimizers.RMSprop` runs slowly on M1/M2 Macs, please use the legacy Keras optimizer instead, located at `tf.keras.optimizers.legacy.RMSprop`.
WARNING:absl:There is a known slowdown when using v2.11+ Keras optimizers on M1/M2 Macs. Falling back to the legacy Keras optimizer, i.e., `tf.keras.optimizers.legacy.RMSprop`.

```
In [45]: image = tf.keras.preprocessing.image.load_img("data/alien-vs-predator/test/alien/99.jpg", target_size=(150, 150, 3))
image = tf.keras.preprocessing.image.img_to_array(image) / 255
image = np.expand_dims(image, axis=0)
prediction = model.predict(image)
```

6. Wnioski





Wnioski

Tabela Podsumowania Alien vs Predator

	Epsilon (ϵ)	Alien									Predator								
	91	92	93	94	95	96	97	98	99	91	92	93	94	95	96	97	98	99	
Model niewyuczony	0 (Oryginalny obraz)	99,89	86,14	99,98	90,33	95,15	99,77	93,53	55,79	99,99	91,99	77,38	99,65	99,92	99,3	56,83	28,66	75,55	99,93
	0,0001	99,89	85,47	99,98	89,25	94,79	99,75	93,17	53,58	99,99	90,8	75,79	99,6	99,9	99,24	55,29	26,52	73,97	99,92
	0,001	99,77	78,63	99,98	75,37	90,47	99,37	89,03	34,6	99,98	72,65	58,24	98,65	99,7	98,55	41,79	12,74	58,55	99,78
	0,01	68,66	27,23	99,71	1,35	26,94	25,83	19,54	0,77	98,14	0,09	0,43	1,41	23,34	43,69	5,33	0,05	2,96	22,58
	0,1	0,05	81,29	33,14	3,27	7,61	20,24	1,98	2,75	0,7	2,49	0,1	0,07	2,53	4,48	7,56	0,02	0,57	1,15
Model wyuczony	0 (Oryginalny obraz)	99,93	91,12	100	96,93	96,47	99,97	96,04	29,89	99,98	96,24	86,23	99,69	99,84	99,55	90,1	46,36	96,47	99,66
	0,0001	99,92	90,52	100	96,48	96,16	99,97	95,79	27,49	99,98	95,44	84,98	99,64	99,82	99,51	89,43	43,04	96,12	99,62
	0,001	99,8	84,11	100	89,88	92,07	99,9	92,89	12,01	99,96	79,54	69,69	98,58	99,26	98,95	81,56	18,51	91,53	98,84
	0,01	52,51	26,06	99,97	6,17	26,67	28,62	29,36	0,11	93,91	0,06	0,38	0,35	1,95	28,8	7,93	0,01	5,22	6,55
	0,1	13,93	99,6	91,83	90,65	80,82	71	89,43	51,12	31,95	0,64	0,07	0	0,17	0,39	1,07	0	0,1	0,22

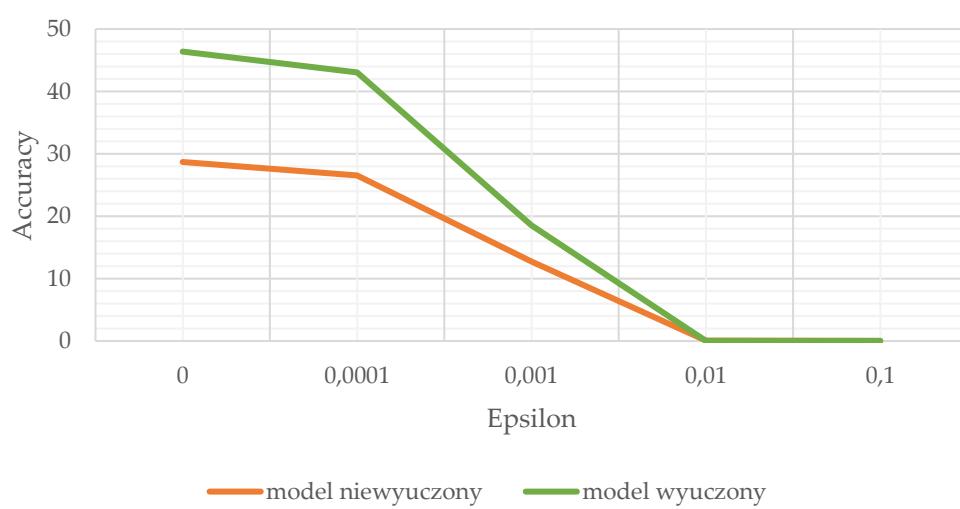
97

PREDATOR

Niskie accuracy – bo zdjęcie źle wykadrowane – za dużo szczegółów

Epsilon	Model niewyuczony	Model wyuczony
0	28,66	46,36
0,0001	26,52	43,04
0,001	12,74	18,51
0,01	0,05	0,01
0,1	0,02	0
0,1	0,07	0

Porównanie dokładności modelu przed i po wyuczeniu





Alien										Predator									
91	92	93	94	95	96	97	98	99	91	92	93	94	95	96	97	98	99		
0,04	4,98	0,02	6,6	1,32	0,2	2,51	-25,9	-0,01	4,25	8,85	0,04	-0,08	0,25	33,27	17,7	20,92	-0,27		
0,03	5,05	0,02	7,23	1,37	0,22	2,62	-26,09	-0,01	4,64	9,19	0,04	-0,08	0,27	34,14	16,52	22,15	-0,3		
0,03	5,48	0,02	14,51	1,6	0,53	3,86	-22,59	-0,02	6,89	11,45	-0,07	-0,44	0,4	39,77	5,77	32,98	-0,94		
-16,15	-1,17	0,26	4,82	-0,27	2,79	9,82	-0,66	-4,23	-0,03	-0,05	-1,06	-21,39	-14,89	2,6	-0,04	2,26	-16,03		
13,88	18,31	58,69	87,38	73,21	50,76	87,45	48,37	31,25	-1,85	-0,03	-0,07	-2,36	-4,09	-6,49	-0,02	-0,47	-0,93		

Wnioski ogólne:

- Kolejna iteracja powinna obejmować większy zbiór danych źródłowych
- Kolejna iteracja powinna zawierać więcej zaszumionych obrazów w stosunku do oryginału (teraz mamy tylko 4)
- Dane źródłowe dla aliena są korzystniejsze niż dla predatora przy dużym zaszumieniu (model po wytrenowaniu i udopornieniu na ataki znacznie lepiej radzi sobie z rozpoznawaniem aliena a zdecydowanie poprawia jakość dla zaszumienia 0,1)
- Model dla aliena przy wysokim zaszumieniu wykazuje progres w przeciwnieństwie do predatora
- Wyniki mogłyby się poprawić, przy dodaniu większej ilości epsilonów